Sustainable Energy, Grids and Networks 26 (2021) 100471

Contents lists available at ScienceDirect

Sustainable Energy, Grids and Networks

journal homepage: www.elsevier.com/locate/segan

Distributed power flow and distributed optimization—Formulation, solution, and open source implementation

Tillmann Mühlpfordt *,[1], Xinliang Dai[1], Alexander Engelmann[2], Veit Hagenmeyer

Institute for Automation and Applied Informatics, Karlsruhe Institute of Technology, Germany

ARTICLE INFO

Article history:
Received 17 November 2020
Received in revised form 16 March 2021
Accepted 19 March 2021
Available online 29 March 2021

Keywords:
Power flow
Distributed optimization
ADMM
ALADIN
MATLAB
Open source

ABSTRACT

Solving the power flow problem in a distributed fashion empowers different grid operators to compute the overall grid state without having to share grid models—this is a practical problem to which industry does not have off-the-shelf answers. We propose two physically consistent problem formulations (a feasibility and a least-squares formulation) amenable to two solution methods from distributed optimization: the Alternating direction method of multipliers (ADMM), and the Augmented Lagrangian based Alternating Direction Inexact Newton method (ALADIN); the latter comes with convergence guarantees. In addition, we provide open source MATLAB code for rapid prototyping for distributed power flow (rapidPF): a fully MATPOWER-compatible software that facilitates the laborious task of formulating power flow problems as distributed optimization problems. Simulation results for systems ranging from 53 buses (with 3 regions) up to 4662 buses (with 5 regions) show that the least-squares formulation solved with ALADIN requires just about half a dozen coordinating steps before the power flow problem is solved.

1. Introduction

The power flow problem is *the* cornerstone problem for power systems analyses: find all (complex) quantities in an AC electrical network in steady state. Planning power systems, expanding and operating them—all of these tasks rest on the power flow problem [1]. Traditionally, transmission system operators (TSOs) and distribution system operators (DSOs) solve and use power flow problems independently of each other, each making modeling assumptions with respect to the other system, e.g. treating the distribution system as a lumped load for the transmission system [2]. Clearly, these modeling assumptions—even if they were valid—may lead to real-world mismatches in both power and voltage. Hence—what is a sovereignty-preserving way to solve power flow problems for large power systems that may be composed of several TSOs and/or DSOs? [3] This is the motivating question of the present paper.

Mathematically, the power flow problem is modeled as a system of nonlinear equations, traditionally solved by Newton–Raphson methods or Gauss–Seidel approaches [1]. These solution techniques may be classified as *centralized approaches*; the full grid model is available to a single central entity. This entity solves the power flow problem, having access to all information not just about the problem itself but also about the solution. Hence, this established approach is in principle able to solve power systems composed of TSOs and DSOs (so-called global power flow problems [2,3])—but only at the cost of giving up sovereignty.

Recently, so-called *distributed approaches* have drawn significant academic attention. These are methods for which several entities (or agents) solve sub-problems independently of each other, then broadcast some—but not all—information to a coordinator [4–8]. The coordinator then solves a coordination problem, and sends to all entities the information they need to solve their sub-problems again.[4] This process is repeated until convergence is achieved. In contrast to centralized methods, decentralized approaches

- distribute the computational effort,
- preserve sovereignty and/or privacy, e.g. grid models,
- decrease the vulnerability due to a single-point-of-failure, and

* Corresponding author.
E-mail address: tillmann.muehlpfordt@kit.edu (T. Mühlpfordt).

[1] The authors acknowledge funding from the German Federal Ministry of Education and Research within the project *MOReNet – Modellierung, Optimierung und Regelung von Netzwerken heterogener Energiesysteme mit volatiler erneuerbarer Energieerzeugung*.

[2] Current address: Institute for Energy Systems, Energy Efficiency and Energy Economics, TU Dortmund, Germany.

[3] Sometimes the literature refers to a power flow problem for a combination of TSOs and DSOs as a *global power flow* problem [2,3].

[4] We clearly distinguish between *distributed* and *decentralized* approaches, the latter requiring no central coordinator whatsoever.

https://doi.org/10.1016/j.segan.2021.100471

- add flexibility.

The interest in distributed approaches is not just academic; there exists a genuine desire by industry to leverage the advantages for real-world problems. In Germany, for example, the horizontal connection between the four TSOs—50 Hertz, TenneT, Amprion, and TransnetBW—is based on centralized power flow solutions. However, new legislation and the undergoing German *Energiewende* toward more renewables force the German TSOs to focus on new vertical cooperation with the numerous DSOs. The so-called *Netzausbaubeschleunigungsgesetz* (grid expansion acceleration bill[5]) introduces the concept of the so-called *Redispatch 2.0* [9]: This *Redispatch 2.0* forces German DSOs to provide accurate day ahead congestions and redispatch measures to alleviate them—all of this in accordance with the four German TSOs by October 2021. Centralized approaches are not favorable for this vertical cooperation mainly due to privacy concerns: the host then combines the role of data owner and product owner, and introduces a possible single-point-of-failure. Hence, it is *not mainly* the distributed computational effort, but more the increase in privacy, reliability, and flexibility that spur the interest of TSOs in distributed approaches.

In the context of *Redispatch 2.0*, especially within the project "DA/RE" (data exchange/redispatch), our industry partner TransnetBW—one of the four German TSOs—explores one possible scenario for *Redispatch 2.0*. Namely, only so-called "light grid models" are exchanged between stakeholders, which are reduced grid models accounting at least for the major grid exchange nodes. The approach we propose in the present paper is slightly different, nevertheless fully compliant with *Redispatch 2.0*: no grid models are exchanged whatsoever between stakeholders, but only voltage information and sensitivities at the coupling nodes.

Large Chinese cities are another example for which the combined power flow problem for TSOs and DSOs is of relevance. In [2] it is argued that many Chinese cities are operating both transmission and distribution systems, both of which are studied and operated separately however. If a computational method were available to solve the combined power flow problem in terms of a privacy-preserving distributed problem, this would be helpful [2,3].

In light of the above considerations the present paper makes the following contributions:

c1: We present two mathematical formulations of distributed power flow problems as privacy-preserving and physically-consistent distributed optimization problems.

c2: We evaluate the applicability of the Alternating direction method of multipliers (ADMM) and the Augmented Lagrangian based Alternating Direction Inexact Newton method (ALADIN) to distributed power flow problems with up to several thousand buses.

c3: We introduce rapidPF: open-source MATLAB code fully compatible with MATPOWER that allows to generate MATPOWER case files for distributed power flow problems tailored to distributed optimization; the code is available on https://github.com/KIT-IAI/rapidPF/ under the BSD-3-clause license.

c4: We extend ALADIN-α, a MATLAB rapid-prototyping toolbox for distributed and decentralized non-convex optimization, to allow for user-defined sensitivities and three new solver interfaces (`fminunc`, `fminunc`, `worhp`).

We explain our contributions relative to the state-of-the-art.

Ad c1: The idea to solve a global power flow problem that stands for the combination of TSOs and DSOs was described in [2,3]. Specifically, [2] coined the term "Master–slave-splitting" to highlight the idea that there is a master system to which several workers are connected[6]; also so-called "boundary systems" are introduced which make up the physical connection between the master and its worker [2]. The solution of the overall power flow problem is obtained iteratively: initialize the boundary voltages, keep them fixed, solve the power flow for the worker systems, then substitute the solution to the boundary system, and solve the master system. This process is executed until the difference of voltage iterates is sufficiently small. Any power flow solver can be used to solve the sub-problems. Hence, the main idea of [2] is to solve the original large power flow problems by breaking it into smaller power flow problems. For each of the smaller power problems, the electrical values for the connection buses have to be fixed, and iterated over. Unfortunately, no convergence guarantees are provided, and the method was applied to systems with less than 200 buses.

In the follow-up work [3] a convergence analysis is carried out, but its practicability is limited due to mathematical settings—such as the implicit function theorem—that are difficult to relate to real-world criteria and/or data. Also, the simulation results from [3] are the ones from [2]. It hence remains unclear how well this method scales. Unfortunately, neither [2] nor [3] provide plots on the actual convergence behavior of their method, or wall-clock simulation times, or the influence of different initial conditions—all of which are aspects relevant to practitioners.

The works [2,3] intertwine problem formulation and problem solution. A consequence of this approach is that [2,3] do not provide a clear and coherent mathematical formulation of the problem they wish to tackle. Also, it is difficult to argue whether the limits in the convergence analysis appear from the problem formulation or the problem solution. Finally, it is difficult to judge whether the problem as they solve is physically consistent. In other words: is the solution according to [2,3] equivalent to the solution of the original power flow problem?

The present paper follows an approach different from [2,3]: we propose to first reformulate the original problem in a truly distributed manner—and then apply method from distributed optimization. We untangle the formulation from the solution, yielding the following advantages:

- clear distinction between problem formulation and problem solution;
- two different mathematical problem formulations that make no assumptions on the sub-problems (e.g. meshed grids vs. radial grids);
- convergence properties follow from theory of distributed optimization;
- reproducible numerical results for test systems with up to $\approx$ 4000 buses.

Ad c2: Recently, distributed optimization techniques have drawn attention for distributed optimal power flow problems. It is especially ADMM that finds widespread application for optimal power flow [4,7,10]. However, ADMM typically requires numerous iterations to approach satisfying numerical accuracy [11]. In addition, ADMM is known to be rather sensitive to both tuning and the choice of initial conditions [12]; line flow limits pose a significant obstacle for ADMM [4]. Furthermore, convergence guarantees for ADMM apply to convex optimization problems, but optimal power flow is known to be non-convex.

[5] Translation by the authors.

[6] We prefer the less inappropriate term "worker" instead of "slave".

There exist distributed optimization methods that are devised for non-convex problems, for instance ALADIN. With ALADIN being a second-order method, it has access to curvature information that speed up convergence, at the expense of having to share more information among the sub-problems. The proof-of-concept applicability of ALADIN to distributed optimal power flow problems has been demonstrated in [8,13,14]; how to reduce the information exchange among the sub-problems is discussed in [15]. ALADIN has more favorable convergence properties than ADMM. That is, within a few dozen iterations ALADIN converges to the optimal solution with satisfying numerical accuracy [8]. Just like with ADMM, however, tuning remains a challenge with ALADIN. Also, the largest test case to which ALADIN was successfully applied is the 300-bus test case.

To summarize: both ADMM and ALADIN have demonstrated their potential for solving distributed optimal power flow problems. It is fair to ask how both methods apply to distributed power flow problems—a question that has not been tackled before to the best of the authors' knowledge. Our findings suggest that ALADIN outperforms ADMM for the distributed power flow problem far more significantly than it does for the optimal power flow problem (in terms of scalability, speed, performance, and tuning).

Ad c3: For academic power system analyses MATPOWER is a mature, well-established, and widely adopted open source collection of MATLAB code [16]. It is not just the many computational facets that MATPOWER provides that make it popular (power flow and several relaxations, optimal power flow, unit commitment, etc.), but also the so-called MATPOWER case file format has inspired other open source packages, for instance PowerModels.jl [17] or PyPSA [18]. Based on both the popularity and the maturity of MATPOWER we provide glue code that solves the following laborious task: given several MATPOWER case files, and given connection information for these case files, construct a MATLAB struct that corresponds to the mathematical problem formulation, and that is amenable to distributed optimization methods. This glue code is called rapidPF, and it is publicly available with a rich documentation—and full MATPOWER compatibility. In addition, rapidPF decreases the time-from-idea-to-result, it computes relevant sensitivities (gradients, Jacobians, Hessians), and it comes with post-processing functionalities. The code for rapidPF is hosted under https://github.com/KIT-IAI/rapidPF/.

The idea of rapidPF is inspired by the MATLAB packages TDNetGen [19] and AutoSynGrid [20]. From a first glance, TDNetGen seems to provide functionality similar to rapidPF. As written in the abstract of [19], TDNetGen is MATLAB code "able to generate synthetic, combined transmission and distribution network models". Unfortunately, TDNetGen is not as flexible as desired: there is currently no straightforward way to generate TDNetGens so-called templates from arbitrary MATPOWER case files. Also, the focus of TDNetGen is on *generating* large test systems, not on *solving* them. In turn, rapidPF allows to both generate test systems and *prepare* them. By preparation we mean the following: to put them in a standard mathematical form amenable to distributed optimization methods such as ADMM and ALADIN. This preparation step must not be underestimated, because providing for an interface to distributed solvers is key in making distributed techniques more popular.

The focus of AutoSynGrid is on generating numerous test systems with similar statistical properties [20]. Hence, the functionality of AutoSynGrid is not directly comparable to either TDNetGen or rapidPF. Nonetheless, the idea of providing the academic community with an open source tool fully aligns with the spirit of rapidPF.

Ad c4: The recently published MATLAB toolbox ALADIN-α provides several implementations of both ADMM and ALADIN [21]. Its user interface allows the user to provide merely the cost functions, the equality constraints, and the inequality constraints. Besides setting several default parameter settings, ALADIN-α computes derivatives required for either ADMM or ALADIN. To do so, ALADIN-α relies internally on CASADI, an automatic differentiation framework that also parses the optimization problem to the low level interface of Ipopt [22,23]. The idea of ALADIN-α is to provide rapid prototyping capabilities for general distributed optimization problems; it is not specifically tailored to distributed (optimal) power flow problems. From the authors' experience, this all-purpose character in combination with CASADI being hard-wired into ALADIN-α hinders it from being applicable to mid- or large-scale power flow problems.

We forked the code and tailored it to the needs of distributed power flow problems: the exact power flow Jacobian is passed from MATPOWER, Hessian approximations are provided, and three solvers are newly interfaced (`fminunc`, `fminunc`, `worhp`).

Although the motivation for the present paper is to solve power flow problems for systems composed of TSOs and DSOs, the authors stress that this setup is not a requirement. The presented methodology is generic in the following sense:

> Given $i \in \{1, \ldots, n^{\text{reg}}\}$ power flow problems, and given suitable connection information, what is a coherent methodology for solving the overall power flow problem in a distributed manner?

It may be that the individual power flow problems happen to coincide with TSOs and/or DSOs, but they can as well be sub-problems of a genuinely large power flow problem that should be solved in a distributed way. In either case, the answer the present paper can provide to the above question is:

> If the distributed power flow problem is formulated as a distributed-least squares problem, and if this problem is solved with ALADIN using a Gauss–Newton Hessian approximation, then the solution is found within half a dozen ALADIN iterations for systems ranging from 53 to 4662 buses.

Remark 1 (*Partitioning*)**.** The present paper assumes that the partitioning of the grid is *given*. For insights on how to partition large grids in computationally advantageous ways, interested readers are kindly referred to [24–26].

The paper is organized as its title suggests: formulation, solution, implementation, followed by an extensive section on results, and concluding comments. The formulation Section 2 introduces nomenclature and the mathematical formulation of the distributed power flow problem. The solution Section 3 covers two methods from distributed optimization: ADMM and ALADIN. The implementation is covered in Section 4, with a strong focus on the open source MATLAB code rapidPF. The results Section 5 gives both qualitative and quantitative assessments of the approach, clearly demonstrating that the least-squares formulation in combination with ALADIN is the most suitable solution approach. Concluding comments in Section 6 close the paper.

2. Problem formulation

Consider a single-phase equivalent of a connected AC electrical network in steady state with $n^{\text{bus}} \in \mathbb{N}$ buses. Solving the power flow problem for this grid means to solve a set of nonlinear equations such that the complex voltage of all buses of the network is found. The standard way to solve power flow problems is to apply a *centralized* method: a single machine determines the solution, for instance, via Gauss–Seidel or Newton–Raphson methods [1]. An alternative is to distribute the computational effort to several

Table 1
List of symbols for distributed power flow.

Symbol	Meaning
n^{reg}	Number of regions
n^{conn}	Number of connecting lines between regions
n_i^{core}	Number of core buses in region i
n_i^{copy}	Number of copy buses in region i
x_i	Electrical state of core buses in region i
z_i	Electrical state of copy buses in region i

machines, leading to so-called *distributed* approaches. Distributed approaches are promising because they eliminate single-point-of-failures, they better preserve privacy, their technical scale-up is easier, and they foster cooperation between transmission and distribution system operators. The idea of *distributed* power flow is to solve *local* power flow problems within each subsystem, independently of each other, and to find consensus on the physical values of the exchanged power between the subsystems, see Fig. 1(a).

2.1. Nomenclature

Before we cover suitable mathematical formulations for distributed power flow, we introduce some nomenclature. For that consider Fig. 1(a), which shows a 12-bus system divided into three subsystems (or so-called *regions*). Suppose we are the operator of region $\mathcal{R}_1 = \{1, 2, 3\}$, for which we know all electrical parameters as well as all bus specifications, and for which we would like to solve a power flow problem. This requires additional information: the complex voltages of buses $\{4, 8\}$, and the branch parameters of the tie lines—hence, connection information about the neighboring subsystems $\mathcal{R}_2 = \{4, \dots, 7\}$ and $\mathcal{R}_3 = \{8, \dots, 12\}$.[7] We shall call buses $\{1, 2, 3\}$ the *core buses* of region $\mathcal{R}_1$, and buses $\{4, 8\}$ the *copy buses* of region $\mathcal{R}_1$; Fig. 1(b) highlights the distinction. The combination of core buses *and* copy buses allows to formulate a self-contained power flow problem for every region.

2.2. Distributed power flow

Table 1 introduces the notation we use from here on: we consider a finite number $i \in \{1, \dots, n^{\text{reg}}\}$ of regions. The (electrical) state x_i of region i contains the voltage angles, the voltage magnitudes, the net active power, and the net reactive power of all *core* buses

$$x_i = \begin{pmatrix} \theta_i^{\text{core}} & v_i^{\text{core}} & p_i^{\text{core}} & q_i^{\text{core}} \end{pmatrix} \in \mathbb{R}^{4n_i^{\text{core}}}. \tag{1}$$

The (electrical) state z_i of region i contains the voltage angles and the voltage magnitudes of all *copy* buses

$$z_i = \begin{pmatrix} \theta_i^{\text{copy}} & v_i^{\text{copy}} \end{pmatrix} \in \mathbb{R}^{2n_i^{\text{copy}}}. \tag{2}$$

Hence, each region i is represented by a total of $4n_i^{\text{core}} + 2n_i^{\text{copy}}$ real numbers. For all core buses of region i the respective $2n_i^{\text{core}}$ power flow equations $g_i^{\text{pf}} \colon \mathbb{R}^{4n_i^{\text{core}}} \times \mathbb{R}^{2n_i^{\text{copy}}} \to \mathbb{R}^{2n_i^{\text{core}}}$, and the respective $2n_i^{\text{core}}$ bus specifications $g_i^{\text{bus}} \colon \mathbb{R}^{4n_i^{\text{core}}} \to \mathbb{R}^{2n_i^{\text{core}}}$ make up the power flow problem for this very region [27].[8] Subtracting the number of equations from the number of decision variables gives a total of

$$\underbrace{4n_i^{\text{core}} + 2n_i^{\text{copy}}}_{\text{Decision vars.}} - \underbrace{2n_i^{\text{core}}}_{\text{Power flow eqns.}} - \underbrace{2n_i^{\text{core}}}_{\text{Bus specs.}} = 2n_i^{\text{copy}} \tag{3}$$

missing equations per region i.[9]

It remains to formalize the information that every *copy* bus from region i corresponds to a *core* bus from a neighboring region $j \neq i$. An example: in Fig. 1(b), bus 4 is a copy bus of region $\mathcal{R}_1$, and it is a core bus of region $\mathcal{R}_2$. Hence, their complex voltage must be identical.

With the above nomenclature the distributed power flow problem reads

$$g_i^{\text{pf}}(x_i, z_i) = 0 \qquad \forall i \in \{1, \dots, n^{\text{reg}}\} \tag{4a}$$

$$g_i^{\text{bus}}(x_i) = 0 \qquad \forall i \in \{1, \dots, n^{\text{reg}}\} \tag{4b}$$

$$\sum_{i=1}^{n^{\text{reg}}} A_i \begin{bmatrix} x_i \\ z_i \end{bmatrix} = 0. \tag{4c}$$

The local power flow problem for region i is given by (4a) and (4b), see Remarks 2 and 3; the so-called consensus matrices $A_i \in \mathbb{R}^{4n^{\text{conn}} \times (4n_i^{\text{core}} + 2n_i^{\text{copy}})}$ enforce equality of the voltage angle and the voltage magnitude at the copy buses and their respective core buses, hence they provide the remaining $4n^{\text{conn}} = \sum_{i=1}^{n^{\text{reg}}} 2n_i^{\text{copy}}$ missing equations, see footnote 9.

Remark 2 (*Power Flow Equations*)**.** The specific form of the regional power flow equations $g_i^{\text{pf}}(\cdot)$ in (4) is arbitrary. Nevertheless, we choose polar coordinates for the voltage phasors when defining the electrical state in (1). In that case, the regional power flow equations are

$$p_j = v_j \sum_{k=1}^{n_i} v_k \left(G_{jk} \cos(\delta_j - \delta_k) + B_{jk} \sin(\delta_j - \delta_k) \right) \tag{5a}$$

$$q_j = v_j \sum_{k=1}^{n_i} v_k \left(G_{jk} \sin(\delta_j - \delta_k) - B_{jk} \cos(\delta_j - \delta_k) \right), \tag{5b}$$

for all buses j from region i; the bus admittance matrix entries are $Y_{jk} = G_{jk} + \mathrm{j}B_{jk}$. For further details we refer to the excellent primer [28].

Remark 3 (*Bus Specifications*)**.** For conventional power flow studies, each bus is modeled as one of the following:

- *Slack bus:* The voltage magnitude and the voltage angle are fixed; the net active and the reactive power are determined by the power flow solution.
- PQ*/load bus:* The active power and the reactive power are fixed; the voltage magnitude and the voltage phasor are determined by the power flow solution.
- PV*/voltage-controlled bus:* The active power and the voltage magnitude are fixed; the reactive power and the voltage angle are determined by the power flow solution.

Mathematically, these are equality constrains of the form $g_i^{\text{bus}}(\cdot)$ for every region i.

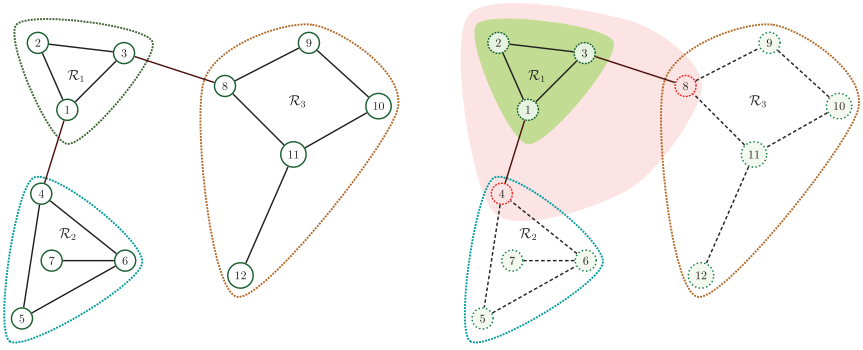
Remark 4 (*Physical Consistence*)**.** The concept of *core buses* and *copy buses* allows to compose the distributed power flow problem in a physically consistent manner: no additional modeling assumptions are introduced or required. If the correct solution to the distributed problem is found, then this will also be the solution to the respective centralized power flow problem. In other words, the concept of *copy buses* and *core buses does not* introduce a structural numerical error [4].

Other approaches, such as "cutting" connecting tie lines and enforcing equality of the electrical state at the intersection [21],

[7] We stress that no information about the *net power* of the neighboring buses $\{4, 8\}$ is required to formulate the power flow equations.

[8] The copy buses are required solely to formulate the power flow equations.

[9] Note that $\sum_{i=1}^{n^{\text{reg}}} n_i^{\text{copy}} \equiv 2n^{\text{conn}}$—we introduce two copy nodes for every line connecting two regions, yielding a total of $4n^{\text{conn}}$ missing equations.



a) Example how to decompose a power grid into three regions $\{1, 2, 3\}$, $\{4, 5, 6, 7\}$, and $\{8, 9, 10, 11, 12\}$.

b) From the perspective of region $\mathcal{R}_1$, the core buses are buses $\{1, 2, 3\}$, and the copy buses are buses $\{4, 8\}$.

Fig. 1. Graphical depiction of nomenclature for distributed power flow problems, see Section 2.1.

are in general *not* physically consistent (only in the absence of line capacitance). Hence, even if the true solution to the distributed problem is found, this solution is not numerically identical to the solution of the centralized power flow problem. In other words, the concept of *cutting lines does* introduce a structural numerical error, generally speaking.

Remark 5 (*Privacy*)**.** To formulate the power flow equations for region i, the voltage information of the copy buses needs to be shared among neighboring regions; this is inherent to the idea of *core* and *copy* buses. Although this means having to share data, the copy bus voltage data (i) does not contain a wealth of privacy information yet (ii) allows for a physically consistent problem formulation, see Remark 4.

2.3. Distributed optimization problem

The distributed power flow problem from (4) is a system of nonlinear equations—in a form amenable to distributed optimization. We propose to solve Problem (4) either as a *distributed feasibility problem*

$$\min_{\substack{x_i, z_i \\ \forall i \in \{1,\dots,n^{\text{reg}}\}}} \quad 0 \quad \text{s.t.} \tag{6a}$$

$$g_i^{\text{pf}}(x_i, z_i) = 0 \tag{6b}$$

$$g_i^{\text{bus}}(x_i) = 0 \tag{6c}$$

$$\sum_{i=1}^{n^{\text{reg}}} A_i \begin{bmatrix} x_i \\ z_i \end{bmatrix} = 0, \tag{6d}$$

or as a *distributed least-squares problem*[10]

$$\min_{\substack{x_i, z_i \\ \forall i \in \{1,\dots,n^{\text{reg}}\}}} \sum_{i=1}^{n^{\text{reg}}} \left\| \begin{bmatrix} g_i^{\text{pf}}(x_i, z_i) \\ g_i^{\text{bus}}(x_i) \end{bmatrix} \right\|^2 \quad \text{s.t.} \quad \sum_{i=1}^{n^{\text{reg}}} A_i \begin{bmatrix} x_i \\ z_i \end{bmatrix} = 0. \tag{7}$$

Necessarily, the solution from the distributed feasibility problem is a solution for the distributed least-squares problem. Both formulations divide and conquer: formulate power flow problems for every region, and relate them by enforcing equal voltages at the connecting buses. The privacy overhead for the regional power flow problems is limited: only the voltage information of the connecting buses is required to formulate the regional power flow equations.

[10] If not stated otherwise, we have $\|\cdot\| \equiv \|\cdot\|_2$.

Table 2
Correspondence of terms from general problem (8) to feasibility problem (6) and to least-squares problem (7).

Term from (8)	Feasibility problem (6)	Least-squares problem (7)
$f_i(\chi_i) =$	0	$\left\| \begin{bmatrix} g^{\text{pf}}{}_i(x_i, z_i) \\ g^{\text{bus}}{}_i(x_i) \end{bmatrix} \right\|^2$
$g_i(\chi_i) =$	$\begin{bmatrix} g^{\text{pf}}{}_i(x_i, z_i) \\ g^{\text{bus}}{}_i(x_i) \end{bmatrix}$	n/a

Both of the given formulations—feasibility (6) and least-squares (7)—are special cases of a general problem formulation. We shall state the general problem formulation in order to simplify the solution algorithms to follow. Using

$$\mathcal{R} = \{1, \dots, n^{\text{reg}}\}, \tag{8a}$$

we define

$$\min_{\substack{\chi_i \\ \forall i \in \mathcal{R}}} \sum_{i \in \mathcal{R}} f_i(\chi_i) \quad \text{s.t.} \tag{8b}$$

$$g_i(\chi_i) = 0 \qquad \forall i \in \mathcal{R} \tag{8c}$$

$$\sum_{i \in \mathcal{R}} A_i \chi_i = 0, \tag{8d}$$

where $\chi_i = (x_i, z_i)$ combines the core bus state and the copy bus state for region i. The consensus constraints (8d) are identical for either problem formulation; the correspondence of the cost and the equality constraints is summarized in the following Table 2.

3. Problem solution

Two viable methods to tackle distributed optimization problems of the form (6) or (7) are ADMM and ALADIN; we provide a brief overview of both. In the following, the superscript ^{k} denotes the k^{th} iterate; the superscript ^{0} hence denotes the initial condition.

Remark 6 (*Wording*)**.** Different problems bring about different wording. In the problem formulation in Section 2 we speak of "regions", because the power flow problem is usually related to an existing physical region. In the problem solution to follow, however, we prefer to speak of "subsystems", and "local problems", because the optimization problems that need to be solved in parallel need not resemble anything that exists in the physical world.

3.1. ADMM

The Alternating direction method of multipliers (ADMM) is a popular method for distributed optimization, particularly for problems in the context of power systems [4,10,29]. Despite the popularity of ADMM, convergence is in general not guaranteed due to the non-convex AC power flow equations. We use ADMM as a benchmark method reflecting the current state-of-the-art for distributed optimization in power systems. There exists a plethora of ADMM variants; the present paper relies on the formulation from [30]. We refer to [11,30,31] for more details on ADMM and its derivations and restrict ourselves to recalling the overall algorithm.

For problem (8), ADMM is summarized in Algorithm 1. In step 1), ADMM solves local optimization problems, where the influence of the neighboring regions is incorporated via auxiliary terms in the objective function considering Lagrange-multiplier estimates $\lambda_i \in \mathbb{R}^{n_i^{\text{conn}}}$ and estimates of the primal variables $\zeta_i^k \in \mathbb{R}^{4n_i^{\text{core}}+2n_i^{\text{copy}}}$.[11] In step 2), all local solutions χ_i^{k+1} are collected in a coordination problem. In many cases, this step can be simplified to a simple averaging step between neighboring subsystems [11]. Finally, the solution of the coordination problem ζ_i^k is sent to each subsystem, and after a local Lagrange multiplier update—step 3)—the iterates start from the beginning.

Its low-weight communication overhead makes ADMM favorable: neighboring subsystems need to exchange but their local solutions, and the coordination step reduces to averaging among neighboring subsystems.

Algorithm 1 ADMM for problem (8)

Initialization: ζ_i^0, λ_i^0 **for all** $i \in \mathcal{R}$, ρ
Repeat:

1) $\chi_i^{k+1} = \underset{g_i(\chi_i)=0}{\operatorname{argmin}} f_i(\chi_i) + \lambda_i^{k\top} A_i \chi_i + \frac{\rho}{2}\|A_i(\chi_i - \zeta_i^k)\|_2^2, \quad i \in \mathcal{R}$ (parallel)
2) $\zeta^{k+1} = \underset{A\zeta=0}{\operatorname{argmin}} \sum_{i\in\mathcal{R}} -\lambda_i^{k\top} A_i \zeta_i + \frac{\rho}{2}\|A_i(\chi_i^{k+1} - \zeta_i)\|_2^2$ (centralized)
3) $\lambda_i^{k+1} = \lambda_i^k + \rho A_i(\chi_i^{k+1} - \zeta_i^{k+1}), \quad i \in \mathcal{R}$ (parallel)

Remark 7 (ADMM *for Non-Convex Problems*)**.** Recently, the convergence of ADMM has been shown for special classes of non-convex problems [32,33]. However, these works consider non-convexities in the objective function, whereas for the AC power flow equations the non-convexity appears in the constraints, for which to the best of our knowledge no convergence guarantee exists so far. Note that divergence of ADMM can occur also for very small-scale problems in the context of power systems [34]; however, this is rarely observed.

3.2. ALADIN

As an alternative to ADMM, the Augmented Lagrangian based Alternating Direction Inexact Newton method (ALADIN) has been proposed [30]. Its main idea is to replace the relatively simple coordination step in ADMM with a more sophisticated one including also constraint and curvature information to yield fast and guaranteed convergence—also for problems with non-convex constraints.

ALADIN for problem (8) is shown in Algorithm 2. Step 1) of ALADIN is similar to ADMM: each subsystem minimizes its objective function with auxiliary terms. A minor difference is that ALADIN maintains one global Lagrange multiplier λ only, and that positive definite weighting matrices Σ_i are considered in the augmentation term, where $\|x\|_\Sigma^2 = x^\top \Sigma x$. In step 2), ALADIN then computes sensitivities of the local problems, i.e. the gradient of the cost function, $\nabla f_i(\chi_i^k)$, an approximation of the Hessian of the Lagrangian function, B_i^k, and the Jacobian matrix of the constraints, $\nabla g_i(\chi_i^k)$. These sensitivities are communicated to a central coordinator, which solves an equality-constrained quadratic program (QP) in step 3) of ALADIN. As a result, primal increments $\Delta\chi_i^k$ are communicated back to the subsystems, which update ζ_i^k and λ^k in step 4), and the iteration starts from the beginning.

In ALADIN, there are two tuning parameters: ν and ρ. The matrices Σ_i can be used for variable scaling—in case of well-behaved problems they can simply be set to the identity matrix. For details on selecting these parameters for optimal power flow problems we refer to [8].

Algorithm 2 ALADIN for problem (8)

Initialization: $\zeta_i^0, \lambda^0, \Sigma_i \succ 0$ **for all** $i \in \mathcal{R}$, ν, ρ
Repeat:

1) Solve for all $i \in \mathcal{R}$
$$\chi_i^k = \underset{g_i(\chi_i)=0}{\operatorname{argmin}} f_i(\chi_i) + \lambda^{k\top} A_i \chi_i + \frac{\nu}{2}\|\chi_i - \zeta_i^k\|_{\Sigma_i}^2, \qquad \text{(parallel)}$$
2) Compute $\nabla f_i(\chi_i^k)$, $B_i^k \approx \nabla_{\chi_i}^2 \left(f_i(\chi_i^k) + \gamma_i^\top g_i(\chi_i^k)\right)$, $\nabla g_i(\chi_i^k)$.
3) Solve the coordination QP
$$\Delta\chi^k = \underset{\Delta\chi}{\operatorname{argmin}} \sum_{i\in\mathcal{R}} \tfrac{1}{2}\Delta\chi_i^\top B_i^k \Delta\chi_i + \nabla f_i^\top(\chi_i^k)\Delta\chi_i \qquad \text{(centralized)}$$
$$+ \lambda^{k\top}\left(\sum_{i\in\mathcal{R}} A_i(\chi_i^k + \Delta\chi_i) - b\right) + \frac{\rho}{2}\|\sum_{i\in\mathcal{R}} A_i(\chi_i^k + \Delta\chi_i) - b\|_2^2$$
$$\text{subject to } \nabla g(\chi^k)\Delta\chi = 0.$$
4) Set $\zeta_i^{k+1} = \chi_i^k + \Delta\chi_i^k$ and $\lambda^{k+1} = \lambda^k + \rho\left(\sum_{i\in\mathcal{R}} A_i\chi_i - b\right)$. (parallel)

Remark 8 (*Choosing Hessian Approximations* B_i^k)**.** ALADIN is guaranteed to converge locally for any positive definite Hessian approximation B_i^k [30]. However, the domain of local convergence and especially the convergence rate depend on the "approximation quality" of B_i^k. Hessian approximations may also reduce the communication and computation overhead, for example via a Broyden–Fletcher–Goldfarb–Shanno (BFGS) approach [8].

Remark 9 (*Communication and Coordination Effort in* ALADIN *and* ADMM)**.** In contrast to ADMM, ALADIN requires more communication and coordination per iteration compared with ADMM. The sensitivities B_i^k and $\nabla g_i(\chi_i^k)$ have to be communicated, whereas ADMM requires to communicate local decision variables χ_i^k only. Also the coordination step in ALADIN is more expensive: instead of computing simple averages, the coordination QP in step 3) of ALADIN requires solving a linear system of equations. With the help of this additional information, however, ALADIN converges faster than ADMM, hence partially compensating for the additional communication overhead. As an alternative to basic ALADIN, which is proposed here, one might consider using bi-level ALADIN [15]. Bi-level ALADIN is able to reduce the per-step communication and coordination overhead even further. We refer to [8,15] for more detailed analytical and numerical comparisons.

4. Implementation

The problem formulations (Section 2) and suggested solutions (Section 3) are moot without means to actually implement, execute, and validate them. We introduce rapidPF, an open source

[11] We deal with non-convex problems, hence the argmin operator in step 1) is to be understood locally.

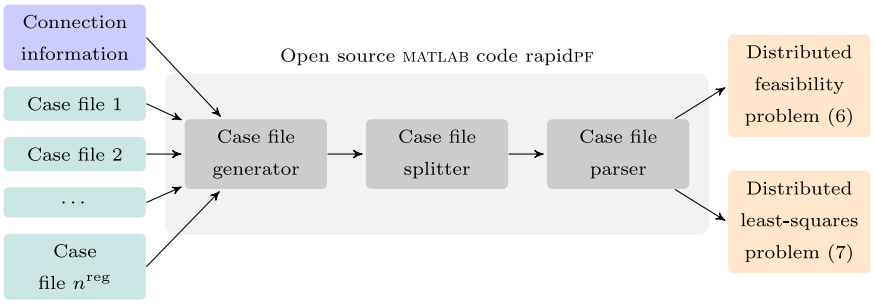


Fig. 2. Flow chart for rapidPF depicting its inputs (case files & connection information) and its output (MATLAB struct compatible with ALADIN-α).

MATLAB code that tackles the problem formulation. Additionally, we present an extension to ALADIN-α, an open source MATLAB code that deals with the problem solution.

4.1. Rapid prototyping for distributed power flow (rapidPF)

Although there exist several excellent open-source tools to model, study, and solve (optimal) power flow problems (e.g. MATPOWER in MATLAB [16], PowerModels in Julia [17], or pandapower in Python [35]), the same cannot be said for *distributed* (optimal) power flow problems—to the best of the authors' knowledge. To help overcome both the tedious, error-prone, and laborious task of formulating distributed power flow problems, and of interfacing distributed optimization methods, we provide open source MATLAB code for rapidPF, which automates the following task:

> Given n^{reg} MATPOWER case files for all $i \in \{1, \dots, n^{\text{reg}}\}$ regions, and given information about how the $i \in \{1, \dots, n^{\text{reg}}\}$ regions are connected, generate a MATLAB struct compatible with ALADIN-α.

The features of rapidPF span:

- *Rapid prototyping:* rapidPF decreases the time-from-idea-to-result.
- *Compatibility:* rapidPF is compatible with MATPOWER and ALADIN-α. All generated case files can be visualized, for example using "Steady-State AC Network Visualization in the Browser"[12].
- *Comparability:* rapidPF generates MATPOWER case files that can be validated by MATPOWER functions such as `runpf()`.
- *Sensitivities:* rapidPF generates function handles for gradients, Jacobians, and Hessians.
- *Documentation:* rapidPF comes with a self-contained and user-friendly online documentation.
- *Open source:* rapidPF is publicly available under the BSD-3-clause license on https://github.com/KIT-IAI/rapidPF/.
- *Post-processing:* rapidPF provides rich post-processing functionalities to analyze the results quickly and intuitively.

The code of rapidPF is made up of three components: the case file generator, the case file splitter, and the case file parser, see Fig. 2. The case file generator requires as inputs several MATPOWER case files in combination with their connection information; the connection information encodes *who* is connected to *whom* and *by what* (kind of branch and/or transformer). The regions can be connected in (almost) arbitrary ways, see Fig. 3.[13] The output of the case file generator is a MATPOWER-compatible merged case file. This merged case file is generated for validation purposes: it provides a reference solution that can be computed, for instance, by running MATPOWER's `runpf()` command. The splitter adds information to each of the n^{reg} case files about its core buses and copy buses. Finally, the parser takes the augmented case files, and generates an ALADIN-α-compatible MATLAB struct that describes the problem either as a distributed feasibility problem (6) or as a distributed least-squares problem (7). The parser also generates sensitivities of the power flow problem, namely the Jacobian of the power flow equations and bus specifications as well as their Hessian information.

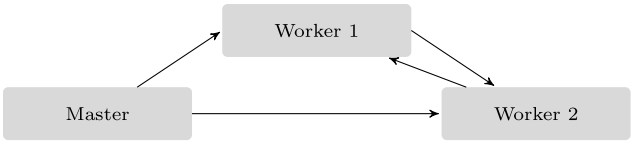


Fig. 3. Supported types of connections between regions.

Remark 10 (*Sensitivities*)**.** All first- and second-order optimization methods require information about derivatives. Hence, rapidPF provides them for the user. The gradient of the local cost function, and the Jacobian of the power flow problem are the exact analytical expressions. The Hessian matrix—required only for ALADIN but not ADMM—is approximated by one of four methods: finite differences, BFGS, limited-memory BFGS, or Gauss–Newton. The first three methods can be applied to both problem formulations (feasibility (6) and least-squares (7)); Gauss–Newton is a method tailored to nonlinear least-squares problems [36], hence applies only to the least-squares formulation (7).

Remark 11 (*Connecting Buses*)**.** A few more words are appropriate about how systems can be connected within the case file generator. First, we formally distinguish between the *master system* and its *worker systems*. The sole difference is that (without loss of generality) the slack bus of the overall system is the slack bus of the master system. The connection between two systems is directed, imposing a natural distinction between the *from*- and *to*-system. For instance, consider the line connecting the *Master* and *Worker 1* in Fig. 3: the *Master* is the *from*-system, *Worker 1* is the *to*-system. The connecting buses in both the *from*- and the *to*-system must be generation buses, hence either a slack bus or a PV bus. If the connecting bus in the *to*-worker-system is the slack bus, then this slack bus is replaced by a PQ bus with zero generation and zero demand. If the connecting bus in the *to*-worker-system is a PV bus, then this PV bus is replaced by a PQ bus with zero generation and its original demand. If no connecting bus in the *to*-worker-system is the slack bus, then the worker system's slack bus is replaced by a PV bus; the respective set points for the active power and the voltage magnitude are taken from the MATPOWER case file entries in `mpc.gen`.

[12] Available on https://immersive.erc.monash.edu/stac/.

[13] The exception being that two buses are allowed to be connected by just one line. Remark 11 provides further guidance about the assumptions on how buses can be connected.

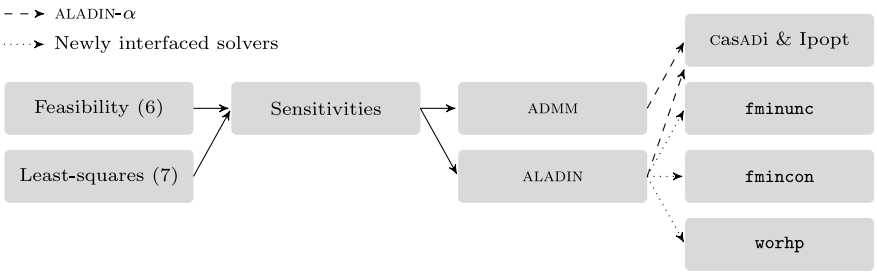


Fig. 4. Problem formulation, problem solution, and interfaced solvers.

4.2. Extensions to ALADIN-α

Whereas rapidPF is MATLAB code tailored to simplify and streamline the *problem formulation*, the open source MATLAB code ALADIN-α is used to tackle the *problem solution* [21]. ALADIN-α provides tested implementations and several variants of both ADMM and ALADIN. Under the hood, ALADIN-α depends largely on CasADi—an open source tool for algorithmic differentiation—and Ipopt as the solver for nonlinear programs. Unfortunately, the sole dependency on CasADi and Ipopt hinders distributed methods from ALADIN-α to be applicable to medium- to large-scale power systems (as we shall discuss in Section 5). Hence, we created a separate branch for ALADIN-α that allows to use the user-defined sensitivities from rapidPF, and that allows to interface different solvers such as `fmincon`, `fminunc`,[14] or `worhp` [37], see also the right-hand side of Fig. 4.

5. Results

We turn to numerical results for power systems of various sizes. We examine several combinations of the two problem formulations—feasibility (6) and least-squares (7)—and the two solution methods—ADMM and ALADIN, paired with different ways to compute sensitivities and interface different solvers, see Fig. 4. The section is devised top-down: we begin with qualitative comparisons of ADMM and ALADIN, then examine the least-squares problem in combination with ALADIN (for different solvers and different Hessian approximations). The final section analyzes the convergence behavior for a 4662-bus system.

Our main finding is that the least-squares formulation with ALADIN and a Gauss–Newton Hessian approximation outperforms all other combinations.

Remark 12 (*Settings Common to All Examples*)**.** For all following examples, the connecting lines between all regions are modeled as transformers with a per-unit reactance of 0.00623, and a tap ratio of 0.985; the resistance, the total line charging susceptance, and the transformer phase shift angle are set to zero.[15]

The initial condition for the primal state (i.e. the state of the electrical grid) is created from the MATPOWER case files as follows: the voltage angle and voltage magnitude are initialized with their respective entries from the entries in the `bus` struct; similarly, the net active power and the net reactive power are initialized as the difference between the respective summed entries in the `gen` struct and the `bus` struct. All dual variables are set to 0.01 initially. Both ALADIN and ADMM terminate if

$$\left\| \sum_{i \in \mathcal{R}} A_i \chi_i \right\|_1 \leq 10^{-10}.$$

[14] The solvers `fmincon` and `fminunc` are part of MATLAB's Optimization Toolbox™.

[15] In light of Remark 6 we switch back to referring to "subsystems" as "regions" and so on.

Table 3
Qualitative comparison of both problem formulations (feasibility vs. least-squares) and their solution by either ADMM or ALADIN.

	Feasibility problem (6)		Least-squares problem (7)	
	ADMM	ALADIN	ADMM	ALADIN
Scalability	poor	acceptable	acceptable	very good
Speed	poor	good	acceptable	very good
Performance	poor	good	acceptable	very good
Tuning	poor	acceptable	acceptable	good

All computed solutions are verified relative to the reference solution provided by the MATPOWER command `runpf()`.

5.1. Qualitative comparison

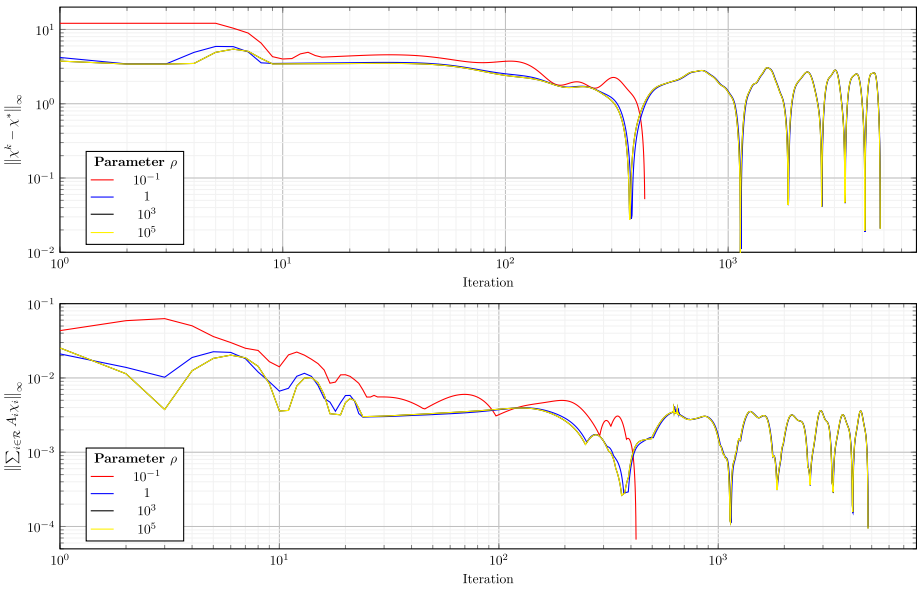
We base our qualitative findings on a total of 7 test cases that are summarized in the first four columns of Table 4. The qualitative comparison comprises both solution methods (ADMM and ALADIN) applied to both problem formulations (feasibility and least-squares).

From Table 3, which summarizes our qualitative findings, it appears that ADMM is unsuitable for either problem formulation. The performance of ADMM depends critically on both the choice of the penalty parameter and the initial condition. Fig. 5(a) shows the convergence behavior for ADMM applied to the feasibility formulation of the 53-bus test case from Table 4. ADMM exhibits strange and overall dissatisfying convergence properties for various choices of the penalty parameter ρ. Most of the considered cases (the ones from Table 4) did not converge successfully even after having done significant parameter sweeps. Fig. 5(a) shows the influence of the choice of the penalty parameter ρ, and the often-encountered convergence behavior with ADMM: after relatively few iterations, the solution is in the vicinity of the optimal solution, but it takes several hundred iterations before further refinement occurs. And even then, the solution is far from being accurate. Fig. 5(b) shows the critical dependence on the (primal) initial condition. Perturbing the primal initial condition around the optimal solution, the plots show that the entire optimization process is prolonged significantly.

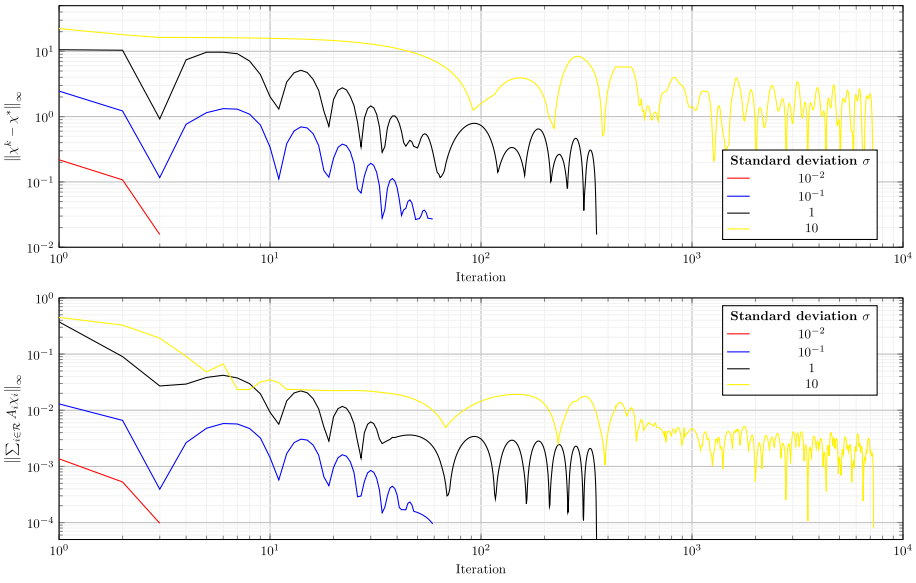
In contrast to ADMM, ALADIN appears applicable to solve the distributed power flow problems from Table 4. In all of the qualitative aspects we consider (scalability, speed, performance, tuning), the least-squares formulation outperforms the feasibility counterpart by far, see Table 3. It is especially the aspect of scalability that hinders the feasibility problem. For instance, the 354-bus test case from Table 4 already took 38.2 s to solve with `fmincon`, and converged within 14 ALADIN iterations.

5.2. Least-squares formulation with ALADIN

Based on our findings from the previous Section 5.1, we consider only the least-squares formulation with ALADIN in the following.



a) Varying penalty parameter $\rho = 10^n$ for $n \in \{-1, 0, 3, 5\}$.



b) Varying intial conditions $\chi^\star + \alpha\hat{\chi}$, where $\chi^\star$ is the true solution, $\hat{\chi}$ is a vector whose entries are samples from a standard normal distribution, and $\sigma \in \{0.01, 0.1, 1, 10\}$ is the standard deviation.

Fig. 5. Convergence behavior of ADMM for a feasibility problem formulation of the 53-bus test case from Table 4. In each subplot, the upper plot shows the distance to the optimal solution, and the lower plot shows the violation of the consensus constraints. See also Remark 12.

5.2.1. Different solvers

We investigate how the different solvers mentioned in Fig. 2 cope with the different test cases from Table 4; we use the sensitivities provided by rapidPF in all cases, i.e. analytical gradients of the cost function, exact Jacobians, and the Gauss–Newton Hessian approximation.

Interestingly, Table 4 suggests that just half a dozen ALADIN iterations are sufficient to solve the test cases, which range from a total of 53 buses to 4662 buses. Hence, the applicability of ALADIN itself is demonstrated. Of course, the overall solution time differs significantly with the choice of the local solver.[16] As a negative result we find that plain ALADIN-α is not suitable for the problem at hand.[17] That is why we chose to implement interfaces for the three other solvers: `fminunc`, `fmincon`, and `worhp`. Although `fminunc` is the seemingly best fit—the local subproblems are unconstrained optimization problems—its practical applicability is limited to subproblems of a few hundred buses. For the 2708- and 4662-bus test systems, `fminunc` takes significantly longer, because the dimension of the local subproblem grows too large. The solution times for `fmincon` and `worhp` are acceptable for all considered cases.

5.2.2. Different hessian approximations

With ALADIN being a second-order optimization method, the (approximated) Hessian. We compare four different Hessian approximations for the least-squares problem (7) with ALADIN: finite

[16] All computations were carried out on a standard a desktop computer with *Intel® Core™ i5-6600K CPU @ 3.50GHz* Processor and 16.0 GB installed RAM; no efforts were made towards parallelization.

[17] By "plain" we mean ALADIN-α that interfaces only CASADI with Ipopt.

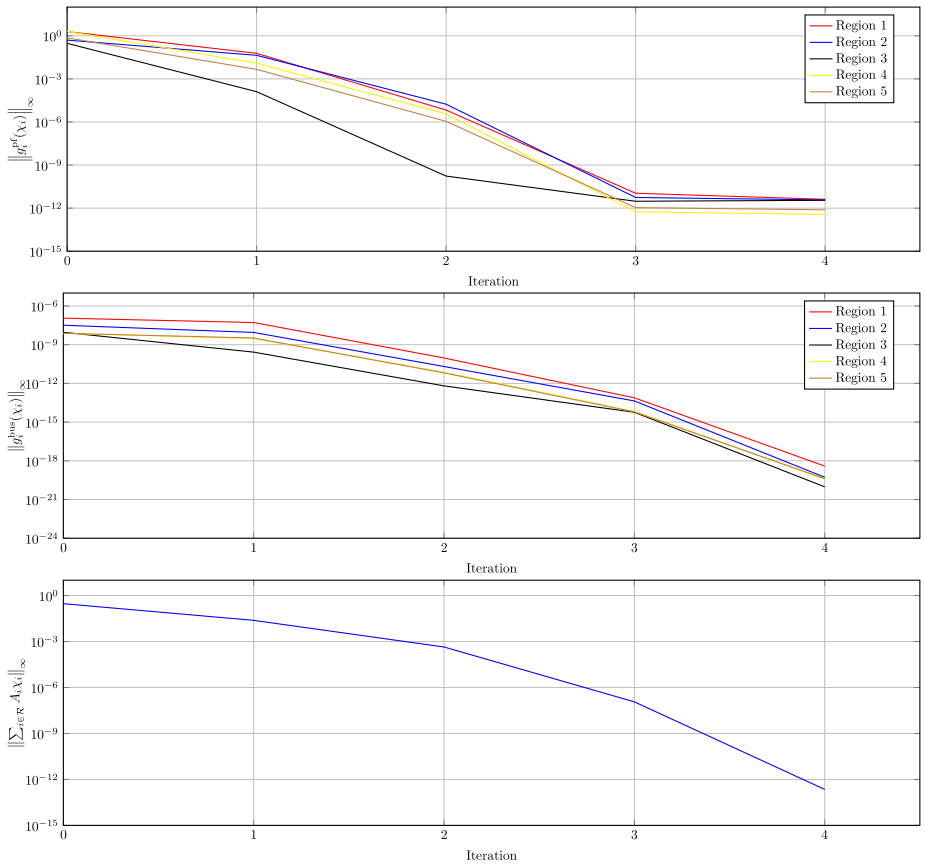


Fig. 6. Decrease of the ∞-norm of the power flow equations, the bus specifications, and the consensus violations, each per ALADIN iteration for the 4662-bus system from Table 4. See also Remark 12.

Table 4
Computing times for different test cases and different solvers when solving the distributed least-squares problem (7) with ALADIN and sensitivities from rapidPF .

Buses	n^{reg}	MATPOWER case files	n^{conn}	Solution time in s for			ALADIN iterations
				`fminunc`	`fmincon`	`worhp`	
53	3	9, 14, 30	3	2.5	2.2	2.4	4
354	3	3 × 118	5	2.5	3.1	4.8	5
418	2	118, 300	2	4.5	5.2	7.0	5
826	7	7 × 118	7	3.7	5.3	7.2	5
1180	10	10 × 118	11	4.9	6.7	9.8	6
2708	2	2 × 1354	1	212.7	41.9	53.6	4
4662	5	3 × 1354, 2 × 300	4	387.9	90.1	113.8	5

differences, BFGS, limited-memory BFGS, and the Gauss–Newton method. The results (see Table 5) confirm: the Gauss–Newton method outperforms all other methods. The finite difference approximation, just like the two BFGS methods, are all-purpose Hessian approximation unaware of the underlying problem structure. Gauss–Newton, in turn, is a Hessian approximation tailored to nonlinear least squares problem, see also Remark 10. The results from Table 5 make it clear that already for small system sizes, the all-purposes Hessian approximations should be avoided.[18]

5.3. 4662-Bus system – Convergence behavior

Next, we study the convergence behavior of the 4662-bus test case. This test case is composed of three 1354-bus MATPOWER test cases, and two 300-bus MATPOWER test cases. Table 6 shows the connecting buses between the regions. For other relevant information such as how the connecting lines are modeled, and how the initial conditions are chosen, see Remark 12.

To solve the distributed power flow problem we choose a least-squares formulation with ALADIN. We use the Gauss–Newton Hessian approximation, and `fmincon` is used to solve the local problems. From Table 4 we see that this setup requires 5 ALADIN iterations and about 90 s. Fig. 6 shows, for every ALADIN iteration and for every region, the ∞-norm of the power flow Eqs. (4a), of the bus specifications (4b), and of the consensus constraint violations (4c). After 5 ALADIN iterations, all violations are below 10^{-10}, and the computations are terminated.

6. Conclusion & outlook

The relevance of distributed power flow problems is increasing, because they allow for better cooperation between different stakeholders, e.g. TSOs and DSOs without exchanging full grid information. Distributed optimization can tackle such distributed

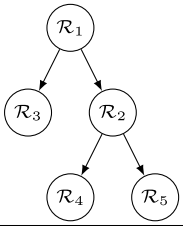
[18] Although they lead to longer computation times, the total number of ALADIN iterations is unaffected.

Table 5
Computing times for least-squares problem (7) with ALADIN and `fmincon`, for different Hessian approximations. The entries in the column "Buses" refers to the entries in Table 4. See also Remark 12.

Buses	Finite difference	BFGS	Limited-memory BFGS	Gauss–Newton
53	10.0	28.6	22.9	2.2
354	61.5	287.8	107.4	3.1
418	185.6	1086.4	148.2	5.2
826	n/a	n/a	n/a	5.3
...	n/a	n/a	n/a	See Table 4
4662	n/a	n/a	n/a	See Table 4

Table 6
Regions and used test cases for 4662-bus test case (left). Connecting buses between regions (middle). Connection graph (right)

Region	MATPOWER case file	From-system		To-system	
		Region	Bus	Region	Bus
1	`case1354pegase`	1	17	2	46
2	`case1354pegase`	1	111	3	271
3	`case1354pegase`	2	64	4	10
4	`case300`	2	837	5	8
5	`case300`				



power flow problems. It is specifically the Augmented Lagrangian based Alternating Direction Inexact Newton method (ALADIN) with its convergence guarantees that yields promising results: if the distributed power flow problem is formulated as a distributed least-squares problem, and if a Gauss–Newton Hessian approximation is used, then about half a dozen iterations suffice to converge to the correct solution. We introduce rapid prototyping for distributed power flow (rapidPF) to facilitate rapid prototyping. It is fully MATPOWER-compatible MATLAB code that takes over the laborious task of creating code amenable to distributed optimization.

Future steps will focus mainly on further structure exploitation for solving the problem, and on implementing larger test cases. The least-squares formulation is promising, hence dedicated nonlinear-least squares techniques should be investigated, e.g. a tailored Gauss–Newton method, or a tailored Levenberg–Marquardt method [36].

The simulation results we presented were all carried out on a single machine. To leverage the literal *distribution* of the optimization, efforts toward parallel computing shall be undertaken when tackling larger test cases. Finally, rapidPF can be extended to optimal power flow problems upon adding cost functions per region.

CRediT authorship contribution statement

Tillmann Mühlpfordt: Conceptualization, Investigation, Methodology, Software, Writing - original draft. **Xinliang Dai:** Software, Investigation, Visualization. **Alexander Engelmann:** Methodology, Writing - original draft. **Veit Hagenmeyer:** Conceptualization, Funding acquisition, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank Jochen Bammert and Tobias Weißbach (both TransnetBW GmbH) for insightful discussions and continuing support and interest in distributed power flow. Finally, Tillmann Mühlpfordt thanks Daniel Bacher for his supporting the migration from Gitlab to GitHub.

References

[1] J. Grainger, W. Stevenson, Power System Analysis, McGraw-Hill Education, 1994.
[2] H. Sun, B. Zhang, Distributed power flow calculation for whole networks including transmission and distribution, in: 2008 IEEE/PES Transmission and Distribution Conference and Exposition, 2008, pp. 1–6.
[3] H. Sun, Q. Guo, B. Zhang, Y. Guo, Z. Li, J. Wang, Master–slave-splitting based distributed global power flow method for integrated transmission and distribution analysis, IEEE Trans. Smart Grid 6 (3) (2015) 1484–1492.
[4] T. Erseghe, Distributed optimal power flow using ADMM, IEEE Trans. Power Syst. 29 (5) (2014) 2370–2380.
[5] B.H. Kim, R. Baldick, Coarse-grained distributed optimal power flow, IEEE Trans. Power Syst. 12 (2) (1997) 932–939.
[6] G. Hug, S. Kar, C. Wu, Consensus + innovations approach for distributed multiagent coordination in a microgrid, IEEE Trans. Smart Grid 6 (4) (2015) 1893–1903.
[7] B. Kim, R. Baldick, A comparison of distributed optimal power flow algorithms, IEEE Trans. Power Syst. 15 (2) (2000) 599–604.
[8] A. Engelmann, Y. Jiang, T. Mühlpfordt, B. Houska, T. Faulwasser, Toward distributed OPF using ALADIN, IEEE Trans. Power Syst. 34 (1) (2019) 584–594.
[9] Bundesministerium der Justiz und für Verbraucherschutz, Netzausbaubeschleunigungsgesetz Übertragungsnetz, 2011, https://www.gesetze-im-internet.de/nabeg/index.html.
[10] J. Guo, G. Hug, O.K. Tonguz, A case for nonconvex distributed optimization in large-scale power systems, IEEE Trans. Power Syst. 32 (5) (2017) 3842–3851.
[11] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, Found. Trends® Mach. Learn. 3 (1) (2011) 1–122.
[12] A.X. Sun, D.T. Phan, S. Ghosh, Fully decentralized AC optimal power flow algorithms, in: 2013 IEEE Power Energy Society General Meeting, 2013, pp. 1–5.
[13] A. Engelmann, T. Mühlpfordt, Y. Jiang, B. Houska, T. Faulwasser, Distributed AC optimal power flow using ALADIN, IFAC-PapersOnLine 50 (1) (2017) 5536–5541, 20th IFAC World Congress.
[14] A. Engelmann, T. Mühlpfordt, Y. Jiang, B. Houska, T. Faulwasser, Distributed stochastic AC optimal power flow based on polynomial chaos expansion, in: IEEE American Control Conference (ACC), 2018, pp. 6188–6193.
[15] A. Engelmann, Y. Jiang, B. Houska, T. Faulwasser, Decomposition of nonconvex optimization via bi-level distributed ALADIN, IEEE Trans. Control Netw. Syst. (2020) 1.
[16] R. Zimmerman, C. Murillo-Sánchez, R. Thomas, MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education, IEEE Trans. Power Syst. 26 (1) (2011) 12–19.
[17] C. Coffrin, R. Bent, K. Sundar, Y. Ng, M. Lubin, PowerModels.jl: An open-source framework for exploring power flow formulations, in: 2018 Power Systems Computation Conference (PSCC), 2018, pp. 1–8.
[18] T. Brown, J. Hörsch, D. Schlachtberger, PyPSA: Python for power system analysis, J. Open Res. Softw. 6 (4) (2018) arXiv:1707.09913.
[19] N. Pilatte, P. Aristidou, G. Hug, TDNetGen: An open-source, parametrizable, large-scale, transmission, and distribution test system, IEEE Syst. J. 13 (1) (2019) 729–737.

[20] H. Sadeghian, Z. Wang, AutoSynGrid: A MATLAB-based toolkit for automatic generation of synthetic power grids, Int. J. Electr. Power Energy Syst. 118 (2020) 105757.
[21] A. Engelmann, Y. Jiang, H. Benner, R. Ou, B. Houska, T. Faulwasser, ALADIN-α – An open-source MATLAB toolbox for distributed non-convex optimization, 2020, arXiv e-prints arXiv:2006.01866.
[22] J. Andersson, J. Gillis, G. Horn, J. Rawlings, M. Diehl, CasADi – A Software framework for nonlinear optimization and optimal control, Math. Program. Comput. 11 (1) (2019) 1–36.
[23] A. Wächter, L. Biegler, On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming, Math. Program. 106 (1) (2006) 25–57.
[24] J. Guo, G. Hug, O.K. Tonguz, Intelligent partitioning in distributed optimization of electric power systems, IEEE Trans. Smart Grid 7 (3) (2016) 1249–1258.
[25] A. Murray, M. Kyesswa, P. Schmurr, H. Çakmak, V. Hagenmeyer, A comparison of partitioning strategies in AC optimal power flow, 2019, arXiv e-prints arXiv:1911.11516.
[26] M. Kyesswa, A. Murray, P. Schmurr, H. Çakmak, U. Kühnapfel, V. Hagenmeyer, Impact of grid partitioning algorithms on combined distributed AC optimal power flow and parallel dynamic power grid simulation, IET Gener. Transm. Distrib. (2020) in print.
[27] S. Frank, S. Rebennack, An introduction to optimal power flow: Theory, formulation, and examples, IIE Trans. 48 (12) (2016) 1172–1197.
[28] S. Frank, S. Rebennack, A Primer on Optimal Power Flow: Theory, Formulation, and Practical Examples, Tech. Rep., Colorado School of Mines, 2012.
[29] E. Dall'Anese, H. Zhu, G.B. Giannakis, Distributed optimal power flow for smart microgrids, IEEE Trans. Smart Grid 4 (3) (2013) 1464–1475.
[30] B. Houska, J. Frasch, M. Diehl, An augmented Lagrangian based algorithm for distributed nonconvex optimization, SIAM J. Optim. 26 (2) (2016) 1101–1127.
[31] D.P. Bertsekas, J.N. Tsitsiklis, Parallel and Distributed Computation: Numerical Methods, vol. 23, Prentice Hall Englewood Cliffs, NJ, 1989.
[32] M. Hong, Z.-Q. Luo, M. Razaviyayn, Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems, SIAM J. Optim. 26 (1) (2016) 337–364.
[33] Y. Wang, W. Yin, J. Zeng, Global convergence of ADMM in nonconvex nonsmooth optimization, J. Sci. Comput. 78 (1) (2019) 29–63.
[34] K. Christakou, D.-C. Tomozei, J.-Y. Le Boudec, M. Paolone, AC OPF in radial distribution networks – Part I: On the limits of the branch flow convexification and the alternating direction method of multipliers, Electr. Power Syst. Res. 143 (2017) 438–450.
[35] L. Thurner, A. Scheidler, F. Schäfer, J. Menke, J. Dollichon, F. Meier, S. Meinecke, M. Braun, pandapower — AN open-source Python tool for convenient modeling, analysis, and optimization of electric power systems, IEEE Trans. Power Syst. 33 (6) (2018) 6510–6521.
[36] J. Nocedal, S. Wright, Numerical Optimization, Springer Science & Business Media, New York, 2006.
[37] R. Kuhlmann, S. Geffken, C. Büskens, WORHP Zen: Parametric sensitivity analysis for the nonlinear programming solver WORHP, in: N. Kliewer, J.F. Ehmke, R. Borndörfer (Eds.), Operations Research Proceedings 2017, Springer International Publishing, 2018, pp. 649–654.